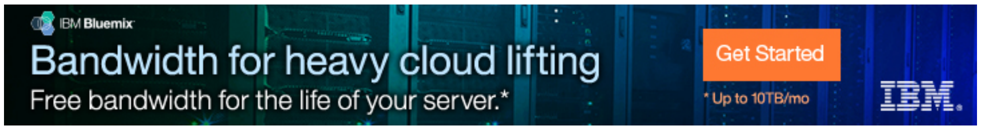
Google FooBar XOR Checksum Challenge


IBM Bluemix
Bandwidth for heavy cloud lifting
Free bandwidth for the life of your server.*
Get Started
* Up to 10TB/mo
IBM.

Google FooBar came up a few days ago and I took it as a challenge to learn python quickly while having fun. However, I ran into one challenge today that has left me stumped, I've come up with a solution that finds the correct result, however it is not fast enough for some test cases. How can I improve upon it?

Queue To Do

You're almost ready to make your move to destroy the LAMBCHOP doomsday device, but the security checkpoints that guard the underlying systems of the LAMBCHOP are going to be a problem. You were able to take one down without tripping any alarms, which is great! Except that as Commander Lambda's assistant, you've learned that the checkpoints are about to come under automated review, which means that your sabotage will be discovered and your cover blown - unless you can trick the automated review system.

To trick the system, you'll need to write a program to return the same security checksum that the guards would have after they would have checked all the workers through. Fortunately, Commander Lambda's desire for efficiency won't allow for hours-long lines, so the checkpoint guards have found ways to quicken the pass-through rate. Instead of checking each and every worker coming through, the guards instead go over everyone in line while noting their security IDs, then allow the line to fill back up. Once they've done that they go over the line again, this time leaving off the last worker. They continue doing this, leaving off one more worker from the line each time but recording the security IDs of those they do check, until they skip the entire line, at which point they XOR the IDs of all the workers they noted into a checksum and then take off for lunch. Fortunately, the workers' orderly nature causes them to always line up in numerical order without any gaps.

For example, if the first worker in line has ID 0 and the security checkpoint line holds three workers, the process would look like this:

```
0 1 2 /
3 4 / 5
6 / 7 8
```

where the guards' XOR (^) checksum is `0^1^2^3^4^6 == 2`.

Likewise, if the first worker has ID 17 and the checkpoint holds four workers, the process would look like:

```
17 18 19 20 /
21 22 23 / 24
25 26 / 27 28
29 / 30 31 32
```

which produces the checksum `17^18^19^20^21^22^23^25^26^29 == 14`.

All worker IDs (including the first worker) are between 0 and 2000000000 inclusive, and the checkpoint line will always be at least 1 worker long.

With this information, write a function answer(start, length) that will cover for the missing security checkpoint by outputting the same checksum the guards would normally submit before lunch. You have just enough time to find out the ID of the first worker to be checked (start) and the length of the line (length) before the automatic review occurs, so your program must generate the proper checksum with just those two values.

Test cases

```
Inputs:
    (int) start = 0
    (int) length = 3
Output:
    (int) 2

Inputs:
    (int) start = 17
    (int) length = 4
```

```
Output:
    (int) 14
```

Solution

Attempt #1

"Let's just do something that works"

The first attempt I had at solving this, it functioned, which was all that I was really hoping for, but was very slow on test cases where length was above a few hundred.

```
def answer(start, length):
  checksum = None
  i = 0
  while length > 0:
    sublength = length
    while sublength > 0:
      if checksum:
        checksum ^= start
      else:
        checksum = start
      start += 1
      sublength -= 1
    length -= 1
    start += i
    i += 1
  return checksum
```

Current Progress

At the moment, I've optimized it to run much faster, I learned about xrange vs range and standard modules I can import. However, even though this can now handle lengths up to 200,000 in a reasonable amount of time, the challenge asks for 2,000,000,000 as a maximum ID, so I can only assume the last challenge has a length close to that. I'm stumped on where to go from here to improve speed, am I looking at this problem fundamentally wrong, maybe I need to find a way to avoid the while loop?

Here's my current iteration:

```
def answer(start, length):
  checksum = reduce(operator.xor, xrange(start, start+length), 0)
  i = 0
  while length > 0:
    start += length + i
    length -= 1
    i += 1
    checksum ^= reduce(operator.xor, xrange(start, start+length), 0)
  return checksum
```

Python is a fun language to work with and I hope to learn from the advice you can provide.

python programming-challenge python-2.7 time-limit-exceeded checksum

edited Dec 9 '16 at 22:00

asked Dec 9 '16 at 19:44
RoflcopterV Roflco
31 5

4 A quadratic algorithm, however optimized, will remain quadratic. To improve speed, you have to consider another algorithm. Check out stackoverflow.com/questions/10670379/... for inspiration. – vnp Dec 9 '16 at 22:39

2 Answers

Let's improve the readability of your second snippet.

You can start by writing your `reduce` only once:

```
def answer(start, length):
    checksum = 0
    i = 0
    while length > 0:
        checksum ^= reduce(operator.xor, xrange(start, start+length), 0)
        start += length + i
        length -= 1
        i += 1
    return checksum
```

And use a `for` loop instead of the while:

```
def answer(start, length):
    checksum = 0
    for i in xrange(length):
        checksum ^= reduce(operator.xor, xrange(start, start+length), 0)
        start += length + i
        length -= 1
    return checksum
```

You can also remove the need for some operations with a decreasing range:

```
def answer(start, length):
    checksum = 0
    for size in xrange(length, 0, -1):
        checksum ^= reduce(operator.xor, xrange(start, start + size), 0)
        start += length
    return checksum
```

And this is as readable as you can get.

You can go up to the one-liner:

```
def answer(start, length):
    return reduce(
        operator.xor,
        (reduce(operator.xor,
                xrange(start + i * length, start + i * length + size),
                0)
         for i, size in enumerate(xrange(length, 0, -1))),
        0)
```

But I would advice against it as it is not readable anymore.

A more interesting change could be to modify the starting ID rather than the length of the queue as we go:

```
def answer(start, length):
    checksum = 0
    for i, begin in enumerate(xrange(start, start + length * length, length)):
        checksum ^= reduce(operator.xor, xrange(begin, begin + length - i), 0)
    return checksum
```

And extract the core functionality into its own function:

```
def xor_in_range(a, b):
    return reduce(operator.xor, xrange(a, b), 0)


def answer(start, length):
    return reduce(
        operator.xor,
        (xor_in_range(begin, begin+length-i)
          for i, begin in enumerate(xrange(start, start + length * length,
length))),
        0)
```

All there is left to do is optimize `xor_in_range` for speed.

edited Dec 10 '16 at 8:16

answered Dec 9 '16 at 22:49
Mathias Ettinger
14.7k 3 15 56

Coco Code Coverage | Free Trial | Cross-Platform Code Coverage Analysis for C/C++, C# and Tcl | froglogic

Even better algorithm with lesser time complexity can be written by making the use of facts

1. each even-odd number pair in that order will XOR to 1
2. And each pair of 1s will XOR to 0.
3. And each 0 results can be ignored because anything XORed with 0 will be 'anything' itself.

Now it is just a matter of figuring out the right combinations for odd and even lengths and odd and even start number.

I won't share the code here because of the confidentiality of Google. But you can check the approach I used here.

https://repl.it/FaCp/3

answered Feb 7 at 12:13
Keyur Golani
109 2

The posted code fails for me, because you are trying to add a `str` to an `int` for example in `result ^=`

`('last_elegible_element_of_that_row' + start)`. – Graipher Feb 7 at 13:33

Graipher, Google makes you accept the terms before giving you the challenge that you'll not post any challenge or its solution online. Hence I've just given high level idea of how to make this optimized. Here the code won't work because I intentionally replaced some part of it. You'll have to come up with the formula to find 'last_element' and 'first_element' from each row and put it at the place for the code to work right – Keyur Golani Feb 8 at 19:45